Remarks

Status of application

Claims 1-48 were examined and stand rejected in view of prior art, as well as stand rejected for certain technical issues. In view of the amendments to the specification and claims and the remarks made herein, reexamination and reconsideration are respectfully requested.

The invention

Applicant's invention comprises a system and methodology for design-time dynamic class type construction. Applicant's invention addresses the problem of efficiently migrating existing programs and tools to a new development framework. For example, the form designer of a development system running on a Microsoft Windows 32-bit operating systems (hereinafter "Win-32") relies on being able to dynamically construct a new form class "on-the-fly" by encoding the class name of the new form class as well as method names, method addresses, field names and field addresses, and the like into tables that are accessible at run-time. By generating these tables "on-the-fly", the layout and structure of a "new" form class can be simulated as it is being constructed by the form designer. A problem arises however in attempting to use this same approach in conjunction with the same development tool on a new development framework (e.g., Microsoft .NET framework) which does not enable a new form class to be constructed on-the-fly in this fashion (i.e., by directly generating and manipulating class metadata as a function of in-memory tables).

Applicant's invention addresses this problem so as to enable existing form design tools to be used on a new development framework. During design of a form, Applicant's solution provides for emitting new meta-data into the design-time process that mimics the type of the class being constructed in the source code, but without actually creating the source code at this point. The solution also includes persistence mechanisms to properly "stream" or persist the meta-data so as to enable the form objects to be recreated at run-time. More particularly, Applicant's methodology for dynamically constructing a form under an object framework includes steps of: providing an ancestor class under an object framework, the ancestor class for representing a form in the development system; in

response to user input, creating a descendant class inheriting from the ancestor class for representing a particular form to be included in the application; generating instructions for creating methods of the descendant class under the object framework; creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application; creating an instance of the descendant class; and constructing the particular form in the development system based on the instance of the descendant class.

General

A. Specification Objections

In accordance with the request of the Examiner, Applicant has amended the specification to include references for several trademarks, including Microsoft, IBM, Java, C#, MacIntosh, Linux, and Solaris when such marks are first referenced in Applicant's specification. However, Applicant takes no position as to the validity or ownership of any such trademarks or whether any such referenced trademarks serve to identify any particular products. In addition, several of the names mentioned by the Examiner are company names (e.g., Novell, Inc., Hewlett-Packard, Dell Computers, Apple Computer) and thus referencing these company names as trademarks would not be appropriate.

Claim Objections

The Examiner objected to claims 2, 28 and 54 as containing informalities. Applicant notes that as Applicant's application does not include a claim 54, the reference to claim 54 is apparently a typographical error. As to claims 2 and 28, Applicant is unclear as to the basis for the Examiner's objection to these claims as the text included in paragraph 3 at page 3 of the office action refers only to claim 29 and does not indicate any particular problems with Applicant's claims 2 or 28.

B. Section 101 Rejection

Claims 25-48 stand rejected under 35 U.S.C. 101 on the basis of non-statutory subject matter. As to these claims, the Examiner states that Applicant's claimed system

constitutes computer programs representing computer listings *per se* and hence are non-statutory. Although Applicant respectfully believes that the Examiner has incorrectly construed Applicant's specification and claims as stating that the elements of Applicant's invention can only be implemented in software, Applicant has amended claim 25 by adding claim limitations of a computer system having a processor and memory. These claim limitations find support in Applicant's specification which expressly states that elements of Applicant's invention may be implemented in hardware, software or firmware (or combinations thereof). This is expressly stated, for example, at [Para 36] of Applicant's specification as follows: "...the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof" (Applicant's specification, [Para 36], emphasis added). Applicant's specification also describes in detail a computer hardware and software environment in which Applicant's invention may be implemented (Applicant's specification, [Para 37]-[Para 48]). As Applicant's claim defines a useful machine or item of manufacture in terms of a hardware or hardware and software combination, Applicant respectfully believes that it defines a statutory product and overcomes the rejection of claims 25-48 under Section 101.

C. Section 112 Rejection

Claims 2 and 26 stand rejected under 35 U.S.C. Section 112, second paragraph as being indefinite. Applicant has amended claims 2 and 26, thereby overcoming the rejection under Section 112, second paragraph.

Prior art rejections

A. Section 103(a): Lam and Albahari

Claims 1-7 and 12-24 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Lam et al, ".NET® Framework Essentials", June 2001, O'Reilly® (hereinafter "Lam") in view of Albahari et al, "C#® in a Nutshell, 2nd Edition, August 2003, O'Reilly® (hereinafter "Albahari"). The Examiner's rejection of Applicant's claim 1 as follows is representative of the Examiner's rejection of these claims as unpatentable over Lam and Albahari:

As to claim 1, Lam discloses in a form-based development system (e.g., Chapter 8 - Windows Forms, Lines 4-6 - how to use Windows Forms .NET® classes to create Windows® Forms-based applications), a method for dynamically constructing a form under an object framework during development of an application by a user, the method comprising:
• providing an ancestor class under an object framework, the ancestor class for representing a form in the development system; in response to user input, creating a descendant class inheriting from the ancestor class for representing a particular form to be included in the application (e.g., Fig. 8-2 - System.Windows.Forms Windows Controls Class Hierarchy; Sec. 8.2 - The System.Windows.Forms Namesapce, 3rd Para. - the System.Windows.Forms namespace provides common set of classes you can use and derive from to build Windows® Forms application; the classes and interfaces in this namespace allow you to construct and render the user-interface elements on a Windows® Form);
• generating instructions for creating methods of the descendant class under the object framework; creating an instance of the descendant class; and constructing the particular form in the development system based on the instance of the descendant class (Sec. 8.3.3 - Visual Inheritance, 2nd Par.- with the advent of Microsoft .NET®, where everything is now object oriented, you can create derived classes by inheriting any base class; since a form in Windows® Forms application is nothing more than derived class of the base Form class, you can actually derive from your form calls to create other for classes).

Lam does not explicitly disclose creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application.
However, in an analogous art of C#® in a Nutshell, Albahari discloses creating a type delegator for the descendant class; thereby enabling the descendant class to track changes made to the particular form during development of the application (e.g., Fig. 35-2 - Exceptions, delegates, and attributes from System.Reflection - element of "TypeDelegator"; Chapter 35. - System.Reflection, 1st Par. - System.Reflection is the API that exposes the full-fidelity metadata of the .NET environment to the programmer; In short, it permits complete access to compile-time data at runtime; Everything is available including fields, methods, constructors, properties, delegate types, and events).
Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Albahari into the Lam's system to further provide creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application in Lam system.
The motivation is that it would further enhance the Lam's system by taking, advancing and/or incorporating Albahari's system which offers significant advantages that reflection offers up a number of possible approaches to user; introspection is the act of using the reflection APIs to discover information about a component assembly (and its constituent types) at runtime without any prior (compile-time) knowledge of it as once suggested by Albahari (e.g., Chapter 35 -

System.Reflection, 2nd Par.).

Under Section 103(a), a patent may not be obtained if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. To establish a prima facie case of obviousness under this section, the Examiner must establish: (1) that there is some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings, (2) that there is a reasonable expectation of success, and (3) that the prior art reference (or references when combined) must teach or suggest all the claim limitations. (See e.g., MPEP 2142). As will be shown, the Lam and Albahari references, even when combined, fail to meet the requisite condition of teaching or suggesting all of Applicant's claim limitations.

At the outset, Applicant does not claim to invented the notion of forms or of source code based construction of an application which includes graphical user interface (window) elements. Applicant acknowledges that it is well known that graphical user interface elements may be described in textual (e.g., source code form) in order to generate a visual representation when the application source code is compiled and placed into production (i.e., at runtime). However, this is not Applicant's invention. Instead, Applicant's invention operates at design time in the context of a development system which is used (e.g., typically by an application developer) for development of an application. Applicant's invention provides for non-source code based construction and manipulation of objects at design-time in a manner which enables the object to appear as having been a source-level construct. Applicant's solution enables a form to be visually designed in a development system without source code or the explicit need to invoke a compiler to generate metadata to represent the form.

Applicant's general approach is as follows. At the time an application is being designed Applicant's invention literally constructs a descendant class that is used to provide a visual representation of the form (i.e., a visual representation of a form that will be included in an application that will run at runtime). The visual representation also enables a user to change and modify elements of the form that will appear as part of the

application at runtime. In other words, the constructed descendant class allows the system to construct at design time the same information that would be available at runtime so that it can be operated on immediately during the design/development phase. It allows, for example, elements of a form to be manipulated on screen in an IDE development environment.

In contrast to Applicant's invention which operates primarily at design time, several of the mechanisms discussed by the Lam and Albahari references operate at runtime after an application has been built, compiled and deployed rather than at design time. For example, the Examiner references Lam for the teaching of providing an ancestor class under an object framework for representing a form in the development system (Lam, Chapter 8, Section 8.2). However, Lam describes source code constructs from which a developer can create a descendant class in the source code. Thus, what Lam is talking about is constructing a textual (source code) representation of what is intended at runtime which can then be compiled in order to generate a runtime version of the form. For example, to selectively hide the Maximize or Minimize button in a control box in Lam's system, the user would set a MaximizeBox or MinimizeBox property to false. As another example, to have the title of a form read "HelloWorld", a user of Lam's system would enter formName.Text = "HelloWorld" (Lam, Section 8.2.2, page 31).

Applicant's invention does not operate in a similar fashion as it does not provide for receiving textual input describing what is intended at runtime. Rather, Applicant's invention actually constructs a visual (in-memory) representation of the form which approximates what the user will actually see when the application being developed operates at runtime (Applicant's specification [Para 60]-[Para 61]). The developer/user can then at design time visually place components (visual design elements) on the form in the user interface of the development system in designing an application (Applicant's specification [Para 57]). Examples of these components include edit controls, buttons, labels, and the like (Applicant's specification [Para 57]). The developer using the form does not need to be aware of the fact that behind the scenes, Applicant's invention is creating these forms by creating and instantiating a descendant class, tracking changes made to the form by the developer and obtaining and persisting sufficient information regarding the form so that the form can be recreated at runtime. Applicant's claims have

been amended in an effort to bring these distinctive features to the forefront. For example, Applicant's amended claim 1 includes the following claim limitations:

> In a form-based development system, a method for dynamically constructing a form under an object framework during development of an application by a user, the method comprising:
> providing an ancestor class under an object framework, the ancestor class for representing a form in the development system;
> in response to user input, creating a descendant class inheriting from the ancestor class for representing a particular form to be included in the application without directly manipulating metadata of the ancestor form;
> generating intermediate language instructions for creating methods of the descendant class under the object framework;
> creating a type delegator for the descendant class, thereby enabling the descendant class to track changes made to the particular form during development of the application;
> creating an instance of the descendant class;
> constructing the particular form in the development system based on the instance of the descendant class and making a design time representation of the form visible to the user without compiling the descendant class;
> tracking changes to the particular form made during development of the application using the type delegator;
> persisting information regarding the particular form; and
> subsequently, generating a version of the particular form at runtime based on the persisted information.
>
> (Applicant's amended claim 1, emphasis added)

As illustrated above, Applicant's invention constructs and displays to the developer a version of the form which mimics the version that will be displayed when the application operates at runtime (Applicant's specification [Para 55]-[Para 57]). This version of the form runs in a limited capacity during the design development phase, enabling the user to visually development the form and tracking and persisting enough information about the form designed by the developer to enable the framework to reconstruct at runtime what was constructed at design time. In response to the developer/user makes changing to the form at design time, Applicant's solution includes a type delegator which tracks the changes being made so that information describing the form that is being designed can be written out and persisted (Applicant's specification [Para 55]-[Para 56]). This information about the form is tracked and persisted so that same form can be reconstructed and look and operate in the manner intended by the

developer at design time (Applicant's specification [Para 57]-[Para 58]).

The Examiner acknowledges that Lam does not disclose creating a type delegator for the descendant class for tracking changes to the form made during development of the application and therefore adds Albahari for these teachings. Applicant's review of Albahari, however, finds that although it discusses use of type delegator, the type delegator operates at runtime (rather than at design time) and is used for purposes different than those of the type delegator of Applicant's claimed invention. As described above, with Applicant's invention the type delegator is used during the design phase of an application to track changes made to a form being designed in a development system prior to compilation of the application. Albahari, in contrast, discusses use of system reflection features to provide access to compile-time data at runtime (Albahari, Chapter 35, System Reflection, 1st paragraph). Thus, Albahari describes use of the type delegator in conjunction with an application operating at runtime to obtain information about the running application. This is not comparable to Applicant's claimed invention which specifically provides that the type delegator operates at design time to track changes to a form being designed.

Moreover, Applicant's invention performs the above-described steps at design time in a manner that does not require actually sending the code being constructed through the compiler (Applicant's specification [Para 53], [Para 61]; see also [Para 55]-[Para 58]). Instead, Applicant's approach essentially provides for constructing the code as a result of what the user is doing at design time for later interpretation by the compiler. In other words, Applicant's solution constructs data in system memory based on user (e.g., developer) input as to what a particular form should look like at runtime. The information is then persisted and used to construct the form and link it up with other portions of the application at runtime (Applicant's specification [Para 55]-[Para 58]). As the application is being built, the design time information is instantiated. It should be noted, however, that there is no compile phase until the user explicitly indicates that they want to compile the application. Albahari, in contrast, uses the type delegator at runtime (i.e., with an already compiled application) to provide access to compile-time data.

All told, neither Lam or Albahari provide for non-source code based construction and manipulation of objects at design-time in a manner which enables the object to

appear as having been a source-level construct. As described above, Lam describes conventional source code construction of user interface elements. In addition, Albahari's type delegator is used in conjunction with an application operating at runtime while Applicant uses the type delegator at design time to track changes to a form being designed without the need to invoke a compiler to generate a representation of the form. Accordingly, as the combined references do not teach or suggest all of the limitations of Applicant's claims, it is respectfully submitted that Applicant's claimed invention is distinguishable over the combined references, and that the rejection under Section 103 is overcome.

C. Second rejection under Section 103: Lam, Albahari and Jazdzewski

Claims 8-11 and 25-48 stand rejected under 35 U.S.C. 103(a) as being unpatentable over Lam (above) in view of Albahari (above), further in view of U.S. Patent 6,002,867 to Jazdzewski (hereinafter "Jazdzewski"). As to these claims, the Examiner continues to rely on Lam and Albahari, but acknowledges that they do not include a number of the teachings included as limitations of Applicant's claims 8-11 and 25-48. The Examiner therefore adds Jazdzewski as providing these teachings.

As to these claims, Applicant believes that the claims are allowable for at least the reasons cited above (as to the first Section 103 rejection) pertaining to the deficiencies of Lam and Albahari as to Applicant's invention. Jazdzewski does not cure these deficiencies. Although Jazdzewski describes visual design of forms in the user interface of a development system, it does not describe the same low-level under-the-hood mechanics of actually achieving that goal as described in Applicant's specification and claims. For example, Jazdzewski makes no mention of using a type delegator to track changes to a form being constructed in a visual design system.

Applicant's invention uses existing facilities of an object framework (e.g., .NET framework) to emit new meta-data into a design-time process that mimics the type of the class being constructed in the source code. It uses the type delegator to track changes and a persistence system to properly "stream" or persist the necessary information used to recreate the objects at run-time as previously described in detail. Jazdzewski and the other prior art references cited by the Examiner, even when combined, do not include

comparable teachings of dynamically constructing a form on-the-fly with out the need for source-code and without having to run commands through a compiler to generate the necessary metadata to enable the form to be reconstructed at runtime. Accordingly, as the combined references do not teach or suggest all of the limitations of Applicant's claims, it is respectfully submitted that Applicant's claimed invention is distinguishable over the combined references, and that the rejection under Section 103 is overcome.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 925 465-0361.

Respectfully submitted,

Date: January 10, 2007

/G. Mack Riddle/

G. Mack Riddle, Reg. No. 55,572
Attorney of Record

925 465-0361
925 465-8143 FAX